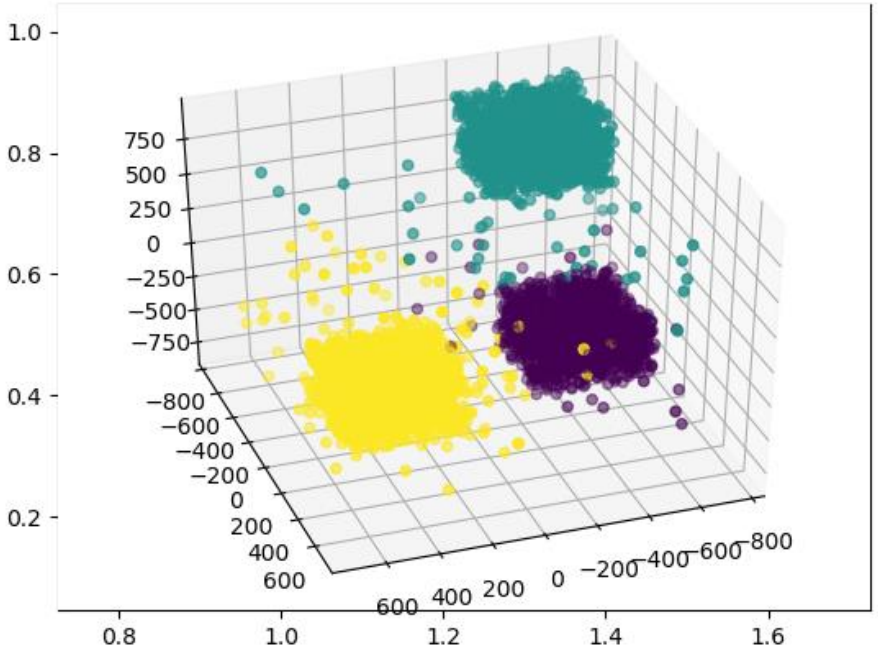


K Means Clustering

Hoàng Hải Hà Trang
Trần Ngọc Hùng Phong

NỘI DUNG

01

TỔNG QUAN

K Means Clustering

K MEANS CLUSTERING

Là một trong những thuật toán cơ bản nhất được sử dụng trong Machine Learning, cụ thể là trong Unsupervised learning

Mục đích: phân chia tập dữ liệu ban đầu thành các cụm khác nhau sao cho dữ liệu trong mỗi cụm có tính chất giống nhau

Một số thuật ngữ được sử dụng trong thuật toán:

- Label: nhãn (của từng điểm dữ liệu)
- Cluster: cụm: tập hợp các điểm gần nhau trong một không gian nào đó
- Center: điểm đại diện của mỗi cluster

BÀI TOÁN

Từ dữ liệu đầu vào và số lượng cluster chúng ta muốn tìm, hãy chỉ ra center của mỗi nhóm và phân các điểm dữ liệu vào các nhóm tương ứng. Giả sử rằng mỗi điểm dữ liệu chỉ thuộc vào đúng một nhóm.

TÓM TẮT THUẬT TOÁN

Đầu vào: Tập dữ liệu X và số lượng cluster cần tìm K

Đầu ra: Các center M và label vector cho từng điểm dữ liệu Y

Các bước:

1) Chọn K điểm bất kì làm các center ban đầu
2) Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất
3) Nếu việc gán dữ liệu ở bước 2 không thay đổi so với vòng lặp trước đó -> dừng lại
4) Cập nhật center cho từng cluster bằng cách lấy trung bình cộng của tất cả các điểm dữ liệu đã được gán vào cluster đó sau bước 2
5) Quay lại bước 2

CÁC HÀM LIÊN QUAN

Loss function (Hàm mất mát):

- Coi center m_k là center của mỗi cluster và ước lượng tất cả các điểm được phân vào cluster này bởi m_k => một điểm dữ liệu x_i được phân vào cluster k có sai số là $x_i - m_k$

=> Mong muốn sai số này có trị tuyệt đối nhỏ nhất: $||x_i - m_k||_2^2$

Vì x_i được phân vào cluster k

$$y_{ik}||x_i - m_k||_2^2 = \sum_{j=1}^{k} y_{ij}||x_i - m_k||_2^2$$

Sai số cho toàn bộ dữ liệu:

$$\mathcal{L}(Y, M) = \sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}||x_i - m_k||_2^2$$

CÁC HÀM LIÊN QUAN

Thuật toán tối ưu hàm mất mát

- Giả sử đã tìm được các centers => tìm các label vector để hàm mất mát đạt giá trị nhỏ nhất
- Giả sử đã tìm được cluster cho từng điểm, hãy tìm center mới cho mỗi cluster để hàm mất mát đạt giá trị nhỏ nhất

02

OPENMP

GIỚI THIỆU

OpenMP là một tập hợp các chỉ thị của trình biên dịch cũng như API dành cho các chương trình được viết bằng C, C++ hoặc FORTRAN cung cấp hỗ trợ cho lập trình song song trong môi trường shared-memory

OpenMP xác định các vùng song song dưới dạng các khối mã có thể chạy song song

CÁC TRÌNH BIÊN DỊCH HỖ TRỢ OPENMP

- gcc: sử dụng –fopenmp
- Clang++: sử dụng –fopenmp
- Solaris Studio: sử dụng –xopenmp
- Intel C compiler: sử dụng –openmp
- Microsoft Visual C++: sử dụng /openmp

OPENMP TRONG C++

- OpenMP bao gồm một tập các chỉ dẫn biên dịch **#pragma** nhằm chỉ dẫn cho chương trình hoạt động
- Các **pragma** được thiết kế nhằm mục đích nếu các trình biên dịch không hỗ trợ thì chương trình vẫn có thể hoạt động bình thường, nhưng sẽ không có bất kỳ tác vụ song song nào được thực hiện như khi sử dụng OpenMP

CÚ PHÁP

- Tất cả các chỉ dẫn OpenMP trong C/C++ đều được dùng thông qua **#pragma omp** theo sau là các thông số và kết thúc bằng một ký hiệu xuống dòng
- **#pragma** chỉ được áp dụng vào đoạn chương trình ngay sau nó, ngoại trừ lệnh **barrier** và **flush**

VÍ DỤ

- **parrallel**
  - *parrallel* bắt đầu một đoạn code thực hiện song song
  - Tạo ra một *team* bao gồm N threads, các lệnh xử lý tiếp theo ngay sau **#pragma** hoặc block tiêp theo (trong giới hạn {...}) sẽ được thực hiện song song. Sau đoạn xử lý này, các thread sẽ được join lại về một
  - Cú pháp: #pragma omp parallel

VÍ DỤ

- **for**
  - Chỉ dẫn *for* sẽ chia vòng lặp for mà mỗi thread trong nhóm thread hiện tại thực hiện một phần của vòng lặp
  - **#pragma omp for** chia các phần của vòng lặp cho các thread khác nhau trong **team**. Mỗi **team** là một nhóm các thread thực hiện chương trình. Khi chương trình bắt đầu, **team** bao gồm duy nhất một thành viên là thread chính đang chạy chương trình
  - Cú pháp: #pragma omp for

03

CÔNG CỤ SỬ DỤNG

Visual Studio Code

PHẦN CỨNG

```
yuufongg@yuufongg:~$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         48 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               AuthenticAMD
  Model name:            AMD Ryzen 5 4600U with Radeon Graphics
    CPU family:          23
    Model:               96
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            1
    Frequency boost:     disabled
    CPU max MHz:         2100.0000
    CPU min MHz:         1400.0000
    BogoMIPS:            4191.82
```

CÔNG CỤ

01

02

NGÔN NGỮ LẬP TRÌNH

01

C/C++

02

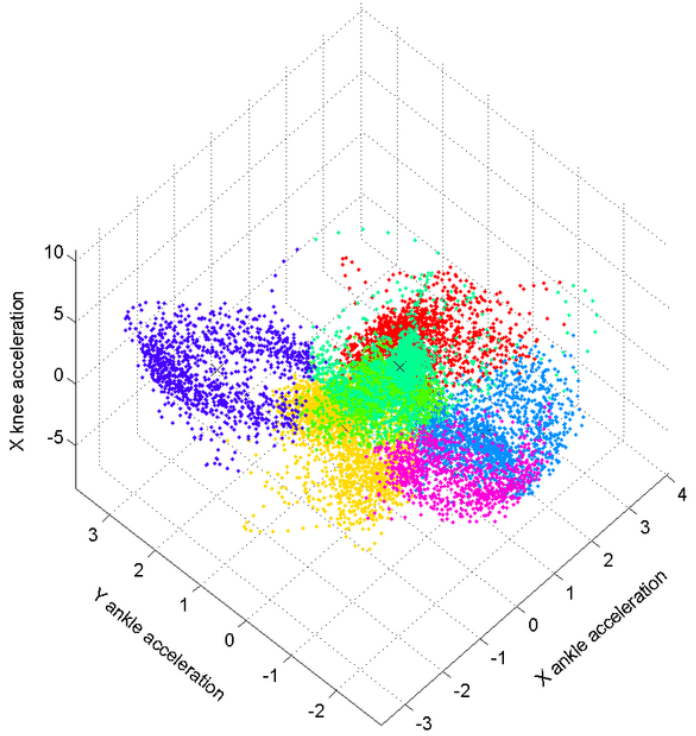
PYTHON

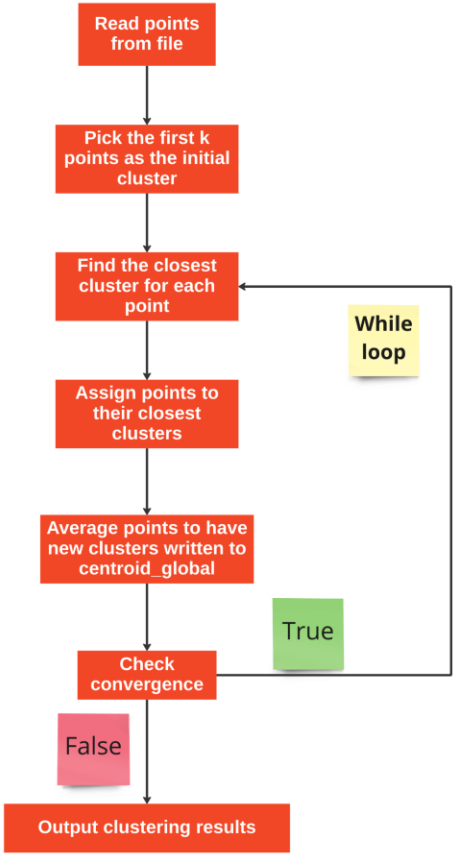
03

LINUX COMMAND

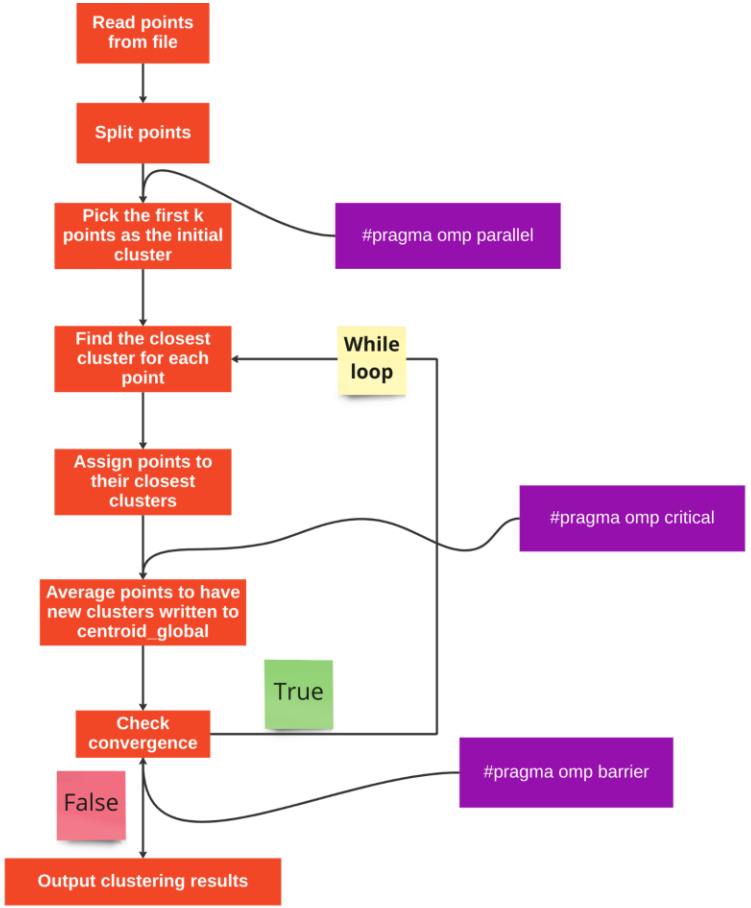
04

STRATEGIES





Tuần tự



Song song hóa

HƯỚNG TIẾP CẬN

- N điểm được chia đều theo số lượng của threads (*num_threads).* Trong trường hợp chia không hết, phần còn lại sẽ được phân bổ vào thread cuối cùng (1)
- **Điểm chính trong thuật toán song song hóa**
  - **Initialization:** K điểm đầu tiên sẽ được chọn làm centroids bắt đầu
  - **Data-parallelism:** Mỗi thread sẽ được phân bổ N/num_threads điểm
  - **Thread function:** Mỗi thread sẽ chạy một vòng lặp (giá trị max_iter là 100). Trong mỗi vòng lặp, mỗi thread sẽ tìm các centroids gần nhất cho mỗi điểm được phân bố, sau đấy phân bố điểm vào các cụm. Sau khi tất cả các điểm được phân bố vào cụm, centroids_global sẽ được cập nhật bằng cách lấy trung bình tọa độ các điểm trong từng centroids
  - **Stopping-condition:** L2-norm được dùng để tính tổng khoảng cách giữa các điểm với centroids của chúng để so sánh với giá trị ở lần iteration trước. Ngoài ra, cũng được so sánh với giá trị threshold (1e-6). Vòng lặp sẽ dừng khi giá trị delta thấp hơn giá trị threshold hoặc hết max_iter.
  - #pragma omp critical: Được dùng cho “thread synchronization", đảm bảo tất cả các thread cập nhật cùng centroids
  - #pragma omp barrier: Được dùng cho “progress synchronization", đảm bảo các thread đều hoàn thành tính toán trước khi chuyển sang step tiếp theo

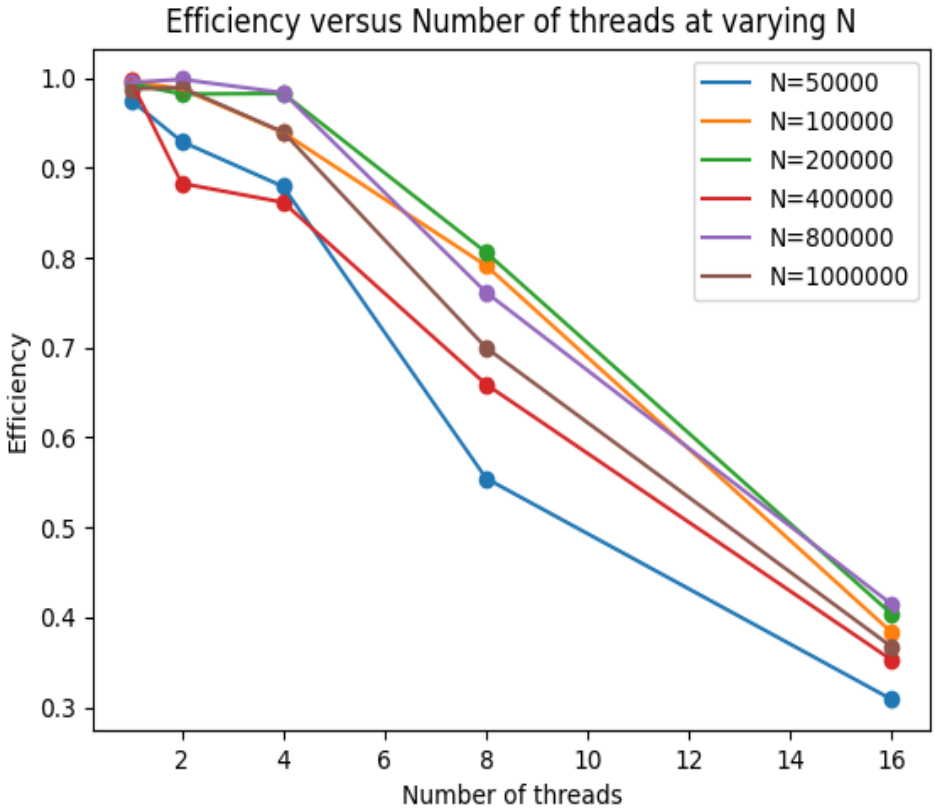
ĐÁNH GIÁ

Efficiency

$$speed\_up = \frac{time\_for\_serial}{time\_for\_parallel}$$

$$efficiency = \frac{speed\_up}{number\_of\_threads}$$

- Kích thước data: `sizes = [50000, 100000, 200000, 400000, 800000, 1000000]`
- Số lượng threads: `threads = [1, 2, 4, 8, 16]`
- Đánh giá: efficiency giảm khi số core càng tăng, dataset càng lớn thì efficiency sẽ cao hơn



ĐÁNH GIÁ

Speedup

$$speed\_up = \frac{time\_for\_serial}{time\_for\_parallel}$$

- Kích thước data: `sizes = [100000]`
- Số lượng threads: `threads = [1, 2, 4, 8, 16]`
  - Đánh giá: số core càng nhiều, tỉ lệ tăng tốc càng lớn.